IAC-02-U.2.05

Ultra Long-Life Spacecraft For Long Duration Space Exploration Missions

Savio N. Chau, Abhijit Sengupta, Tuan A. Tran, Alireza Bakhshi
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, USA

53rd International Astronautical Congress
The World Space Congress - 2002
10-19 Oct 2002/Houston, Texas

ULTRA LONG-LIFE SPACECRAFT FOR LONG DURATION SPACE EXPLORATION MISSIONS

Savio N. Chau — Principal Engineer
Abhijit Sengupta — Senior Engineer
Tuan A. Tran — Senior Engineer
Alireza Bakhshi — Senior Engineer
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, California, USA
Savio.n.chau@jpl.nasa.gov

ABSTRACT

The predominant failure mode in an ultra long-life system is the wear-out of components. In order to survive long duration missions, current fault tolerant design techniques would require excessive number of redundant components. This paper describes a more efficient fault-tolerant avionics system architecture that requires much less redundant components. This architecture employs generic function blocks that can be programmed to replace a wide variety of components in-flight. Hence, each individual generic block is essentially equivalent to almost an entire redundant string of components in the conventional approach. In that way, the ultra long-life system can achieve much higher level of reliability while carrying much less components. On the other hand, due to the programmability of the generic redundant blocks, the physical location of a specific component might not be pre-determined. Therefore, wireless interconnection is employed to provide the necessary flexibility in connectivity. A testbed of this architecture is being developed at the Jet Propulsion Laboratory.

INTRODUCTION

After decades of Solar System exploration, NASA is close to completion of the initial reconnaissance, and has began landing and sample return missions on many planets, satellites, comets, and asteroids. The next logical step for space exploration is to expand the frontier to the far reaches and beyond the solar system. Possible missions include Pluto and Kuiper Belt objects sample return or interstellar space exploration. These missions can easily last for more than 30 to 50 years.

In addition to deep space exploration, many terrestrial spacecrafts also require ultra long life design. One of the major factors in the maintenance cost of communication satellite networks is the replacement of failed spacecrafts. The total cost for manufacturing, testing, and launch to replace a failed satellite exceeds hundreds of millions of dollars. In addition, there are also costs associated with the lost of services during a down time.

Ultra long-life space systems need breakthrough technologies in four main areas [1]:

- long-term survivability – to handle failures due to random events, design errors, and wear-out mechanisms;
- optimal management (administration) of consumable resources – to maximize the acquisition and minimize the consumption of consumable resources such as power, fuel, and the generic blocks;
- evolvability and adaptability – to have built-in mechanisms so that the capabilities and functions of the spacecraft can be updated after launch; otherwise, the useful life of the spacecraft will be limited by the obsolescence of the on-board technology, and
- long-term operation of the spacecraft – to reduce the operation costs and maintain a workforce knowledgeable of the spacecraft.

The following discussion will focus on the architectural aspect of the long-term survivability.

LIMITATION OF CONVENTIONAL FAULT TOLERANT AVIONICS ARCHITECTURES

The current technologies and spacecraft design techniques are not adequate to support a 50-year mission. While all spacecraft avionics architectures have fault tolerance, they are designed to tolerate random failures, which can be handled effectively by dual or triple redundancy for a relatively short mission life. However, they are not suitable for long duration missions since they are not very efficient in utilizing redundant components. For example, in dual-string architecture, the system fails when the processors in both strings have failed. It is impossible to use the other components in the system to resurrect the system even though they are still functioning properly. Consequently, the improvement of system reliability by the conventional fault tolerance architecture will diminish in time. This can be illustrated by reliability modeling as shown in Figure 1.

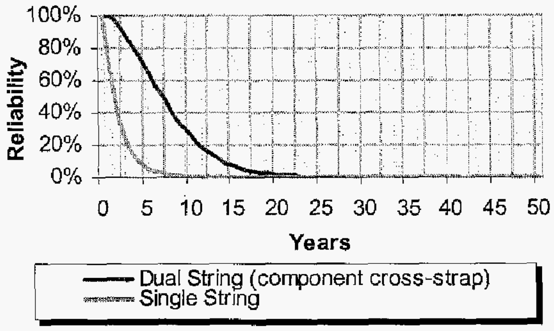

Figure 1: Reliability of Single and Dual String Systems

Furthermore, the predominant failure mode in an ultra long-life system is the wear-out of components. All active components in the system are destined to fail before the end of the mission. Therefore, the conventional fault tolerant architecture would require many more redundant components, as they are not efficient in utilizing the redundant components. An ultra long life missions would require much more efficient fault tolerant architecture to reduce the number of redundant components required.

AN AVIONICS ARCHITECTURE FOR ULTRA LONG LIFE MISSIONS

The NASA Exploration Team has developed a highly reconfigurable avionics system architecture that uses redundant resources much more efficiently. This architecture employs *Generic Function Blocks* that can be configured or programmed in-flight. This is illustrated in Figure 2.

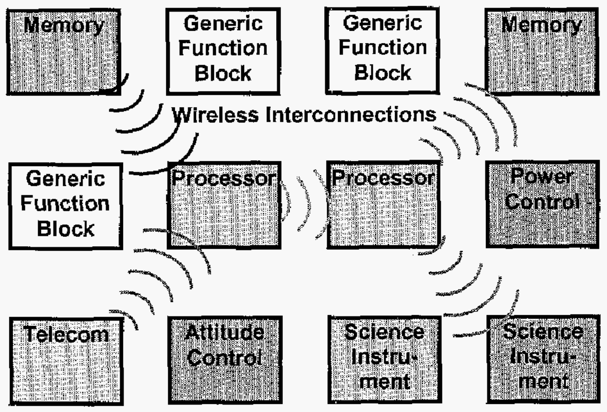

Figure 2: Avionics System with Generic Function Blocks

When any function block fails, one of the *Generic Function Blocks* will be configured to replace the failed block. Hence, in some sense, each individual generic block is almost equivalent to an entire redundant string of components in the conventional approach. For this reason, the ultra long-life system can achieve much higher level of reliability while carrying much less redundant components.

A reliability model shown in Figure 3 illustrates the reliability of the fault-tolerant architecture using Generic Function Blocks. It compares a dual-string system, in which each string has 8 components, to an ultra long life avionics architecture consisted of 16 *Generic Function Blocks*, assuming the Mean-Time-Between-Failure of each component is 125,000 hours (14 years). It is clear that the *Generic Function Block* architecture has much higher reliability than the dual-string system.

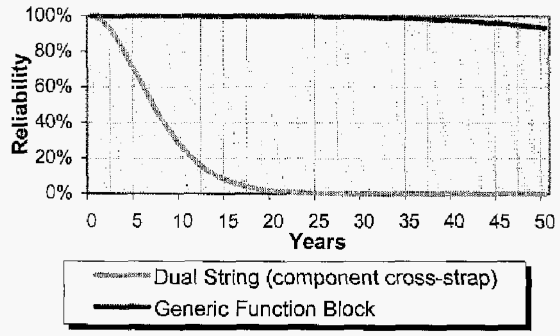

Figure 3: Reliability of Dual String and Generic Function Block Systems

Providing connectivity among the generic blocks is an obstacle in the implementation of the *Generic Function Blocks* architecture as the needed connectivity will be difficult to determine before launch. This is because a *Generic Function Block* should have the capability to replace any one of the components, regardless of their relative positions. This may result in a drastically different configuration after the fault recovery. Therefore, the connectivity among components has to be very flexible. A conventional solution is to use a switched network connection such as a crossbar switch. However, the complexity of a switched network can grow very rapidly with the number of functional blocks in the system.

The approach taken in this research is to use wireless interconnection to replace the switched network (see Figure 2). Since the signals of a wireless network are broadcast, it is not constrained by the physical locations of the function blocks as long as the distance between two blocks is within the broadcast range. This usually is not a problem for avionics systems since all the components are confined in small space in most cases.

FAULT DETECTION IN THE ULTRA LONG LIFE AVIONICS SYSTEMS

Since the components in the *Generic Function Block* architecture are not duplicated, it is more difficult to use duplicate-and-compare or voting to detect failures. For some failure modes such as data corruptions, error detection and correction codes are still applicable. On the other hand, function block level failure modes such as crash cannot be detected as directly as the duplicate-and-compare method. The conventional method to solve the problem is to use watchdog timers. However, the detection latency of a watchdog timer is unacceptable in many applications.

The approach this research has taken is to use the Autonomous Testing [2][3][4]. Autonomous Testing is a distributed fault detection technique, in which each component in the system is tested by several other components and the identity of failed components are derived from the test results. As an example, a *Generic Function Block* architecture with three function blocks, as shown in Figure 4, might be considered. Periodically, each function block in Figure 4 is tested by itself and another block as follows.

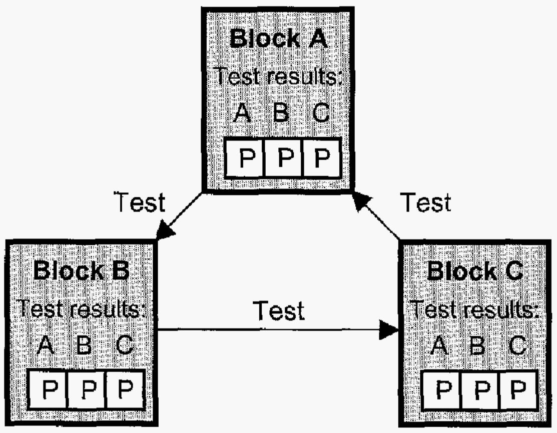

Figure 4: Autonomous Testing System Example

First, each node will perform a self-test. In a fault free situation, each block should pass the self-test. This is indicated in the diagram that each block has a P (i.e., pass) in the *Test Result Register* under its own name.
Second, in the test phase, each block is tested by its testers (for the example in figure 4, each block is tested by another block). For this example, the block A tests the block B, the block B tests the block C, and the block C tests the block A. If there is no fault, each block should also pass each test. This is indicated in the diagram that each block has a P in the *Test Result Register* under the name of the block it has tested.
Third, in the diagnostic phase, each block *x* forms its opinion about the health status of any other functional block *y* either from the result of testing, if *x* tested *y*, or from the opinion of some other block *z* such that *z* is healthy in *x*'s opinion. Hence, even though the block A has not tested the block C, but by the opinion of the block B (whom the block A finds healthy), the block A finds that the block C is also healthy and uses this information to update its *Test Result Register* in the host computer accordingly.

When one of the functional blocks fails, its upstream neighbor will detect its failure and that information will propagate to the other functional blocks in the diagnostic phase, so that the test results in the failed block will be ignored. And it has been proven that all the healthy functional blocks in the system can deduce a consistent diagnosis from the test results about which function block(s) have failed. This is illustrated in Figure 5, where the block B has failed. In the test phase, the block A detects the block B's failure and the block C finds out that the block A is healthy. In the diagnostic phase, the block C finds out from the block A that the block B has failed. On the other hand, the block A finds the block B faulty and ignores its opinion about other block(s) – in this case the block C. However, since it is assumed that there is only one failure per processing cycle, and since the block A already knows that the block B has failed, therefore the block A can determine that the block C is healthy by deduction. Therefore, both the blocks A and C have a consistent diagnostic that the block B has failed, so that the block B will be ignored in subsequent operations. A more general study of the Autonomous Testing can be found in [2-4].

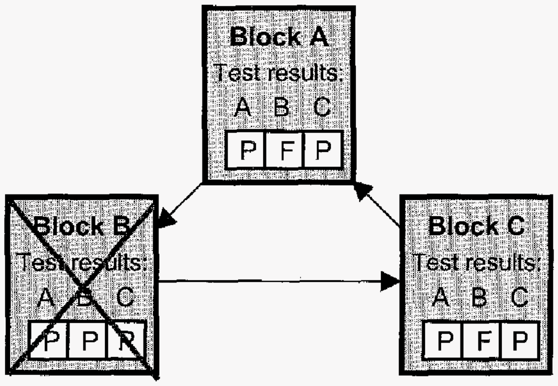

Figure 5: Diagnostic with Autonomous Testing

ULTRA LONG LIFE AVIONICS ARCHITECTURE PROTOTYPE

A prototype of the ultra long life avionics architecture is being developed at the Jet Propulsion Laboratory. This prototype is intended to demonstrate the three key aspects of the architecture: (1) the feasibility of fault recovery with generic function blocks, (2) the feasibility of system reconfiguration with wireless interconnections, and (3) fault diagnosis with distributed autonomous detection.

The "system" that this prototype implements is a navigator that has a gyroscope, star tracker, and accelerometer. Each of the sensors has its own controller, which collect data from the sensor and send it to a local controller for processing. The conceptual design of the prototype is shown in Figure 6.

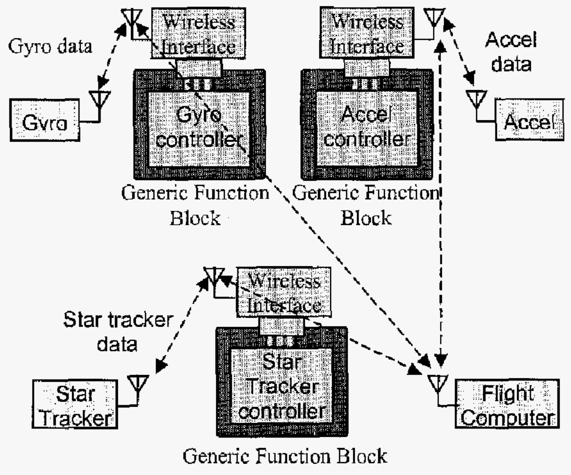

Figure 6: Navigator implemented by Generic Function Block

A Generic Function Block implements each sensor's controller in the prototype. Also an 8051 micro-controller is implemented within the FPGA. The *Generic Function Block* contains an FPGA with 600 thousand gates, a wireless interconnection interface, a 64 Kbytes of program memory, and other supporting circuits and displays.

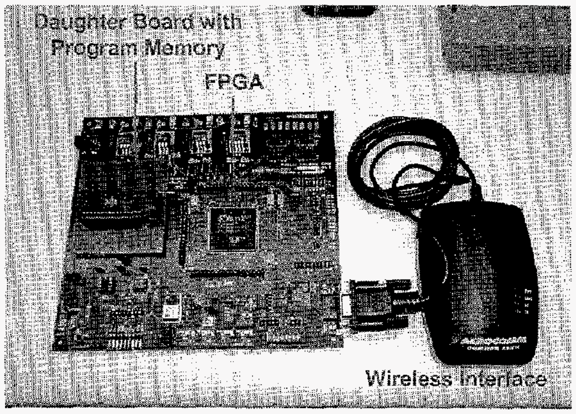

Figure 7: Generic Function Block Implementation

The wireless interconnection is a commercial proprietary protocol, similar to IEEE 802.11b standard [5]. An implementation of the *Generic Function Block* is shown in the Figure 7.

There are three types of software running on the *Generic Function Block*: navigation sensor interface function, autonomous testing, and fault recovery and reconfiguration. Each *Generic Function Block* has the software for all sensor interface controllers but is assigned to run only one type of sensor control software at the system initialization. The autonomous testing software executes the autonomous testing. The testing performed is a simple reading of a watchdog timer from the *Generic Function Block* under test. The *Generic Function Block* can deduce which block has failed by examining the testing results as described in last section. When a block fails, its function will be assigned to its upstream block neighbor. This is the responsibility of the fault recovery and reconfiguration software.

Due to resource limitations few simplifications are needed to enable the implementation of the prototype in a timely manner. First, since there are no available sensors equipped with the wireless interface, a host computer is used to simulate these sensors. The host computer sends the sensor data to the corresponding interface controller by broadcasting the data along with the sensor name (in form of an address). Only the *Generic Function Block* that has the correct sensor controller will accept and respond to command and data.

Another simplification in this implementation was to use the host computer as a pass-through channel and arbitrator for the communications among the *Generic Function Blocks*. In other words, when a *Generic Function Block* sends a message to another block, it first sends the message to the host computer, which then re-broadcasts the message so that the function block with the correct destination address will receive and respond to the message. The host computer also sets up the *Generic Function Blocks* such that no more than one block will send message at any time. This simplification alleviates the prototype from worrying about the details of the arbitration protocol, so that more effort can be focused on the design of the fault recovery and system reconfigurations. However, this simplification will be removed from future prototypes.

The full testbed with all *Generic Function Blocks* and the host computer is shown in Figure 8. The system integration and test are still underway. When the prototype is competed, the fault recovery with *Generic Function Blocks* can be demonstrated by injecting a fault into one of the blocks (e.g., turning off the power). Then, it is expected that the failure will be detected by the autonomous testing and the task on the failed block will be assumed by its upstream neighbor.

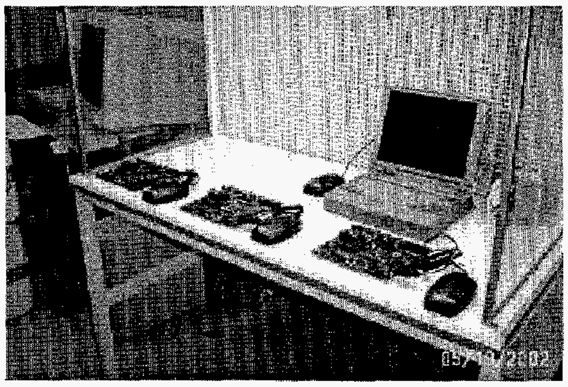

Figure 8: Ultra Long Life Avionics Architecture Testbed

EXPERIMENT OF HARDWARE RECONFIGURATION WITHIN A GENERIC FUNCTION BLOCK

In the prototype, the system reconfiguration and reallocation of functions from one block to another is basically achieved by software. This is possible in the prototype because all the sensor interface controllers have very similar designs. However, in a more general case, blank *Generic Function Blocks* have to be programmed to replace a failed function block. In that case, hardware reconfiguration will be necessary.

The Ultra Long Life Avionics research team at the Jet Propulsion Laboratory has also conducted an experiment to reconfigure *Generic Function Block* through wireless inter-connection. A circuit board was constructed for this experiment as shown in Figure 9. This circuit board also contains an FPGA and a number of I/O interfaces. One of the I/O interface, a parallel port, is modified so that it can accept the configuration for the FPGA from a computer through wireless interconnection. The testbed for the hardware reconfiguration experiment is shown in Figure 10.

The FPGA contains three simple interface circuits: an LED interface, an LCD interface, and a switch interface. In addition, it also dedicates an area of the FPGA as "spare logic" that can be reprogrammed to replace either the LED or the Switch Interface. The circuit board also includes two sets of switches for fault injection into the LED and Switch Interface circuits. These two interface circuits detect the injected faults by monitoring the positions of these switches. The design of the circuit board and FPGA is depicted in Figure 11.

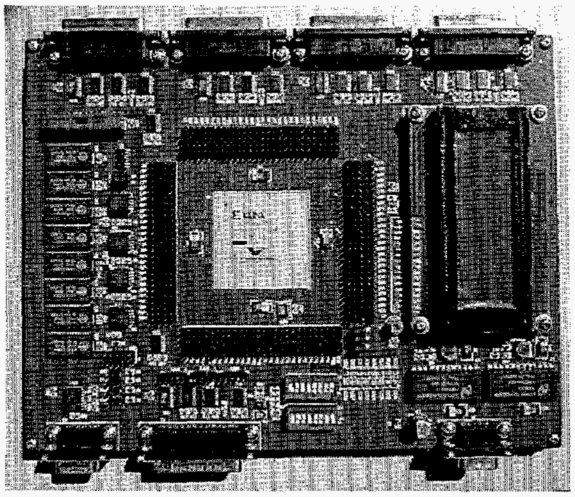
Figure 9: Hardware Reconfiguration Demonstration Circuit Board Design

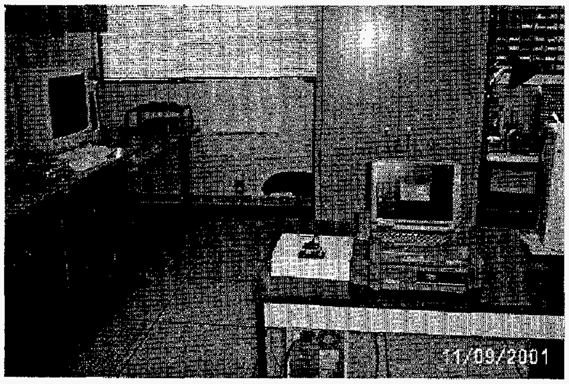

Figure 10: Hardware Reconfiguration Demonstration Testbed

In this experiment, the configuration of the FPGA was first downloaded from the computer to the chip through the wireless interconnection. In normal operation, the System Switches (Figure 11) could be set such that the LED interface could turn on an array of LEDs in different patterns. When a fault was injected into the LED interface, the LEDs would not be turned on properly and an error status signal was sent back to the computer through the wireless interface. Upon receiving the error status, the computer downloaded a new configuration file to the FPGA, again through the wireless interface, so that the spare logic was used to replace the LED interface. Similarly, when a fault was injected to the Switch Interface (Figure 11), the System Switches would not function properly and an error status was sent to the computer. Consequently, a new configuration file was downloaded to the FPGA, so that the spare logic was used to replace the Switch Interface. This experiment was demonstrated successfully in the testbed.

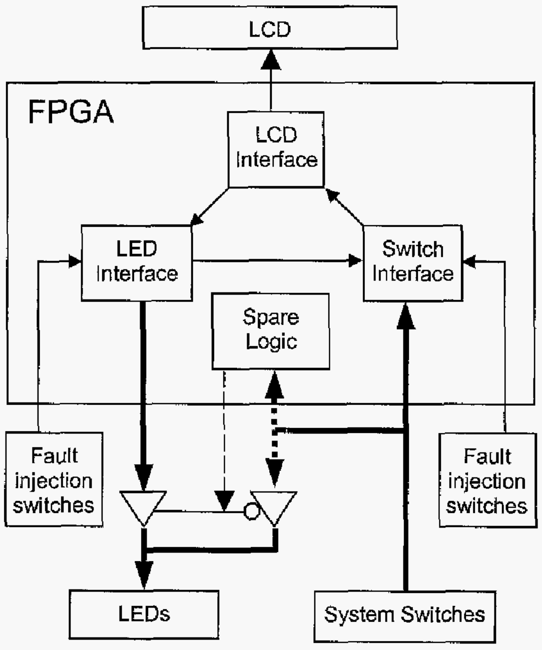

Figure 11: FPGA Design for Hardware Reconfiguration Demonstration

This hardware reconfiguration capability will be incorporated in future Ultra Long Life Avionics experiments.

CONCLUSION

This paper has explored some unconventional architecture design techniques that utilize much less redundant components for sustaining very long duration missions. An architecture based on the concept of *Generic Function Blocks* has been developed. A prototype of this architecture representing a navigator subsystem is being constructed at the Jet Propulsion Laboratory. This prototype can detect failures in any one of the *Generic Function Blocks* by means of Autonomous Testing through wireless communication among the blocks. Once the failed block is identified, software technique is employed to relocate the tasks of the failed block to a healthy block.

An independent experiment of reconfiguring the hardware design of an FPGA through wireless interconnection has also been conducted. The experiment was conducted successfully and the technique will be incorporated into future system reconfiguration/recovery experiments.

FUTURE WORK

Many issues of this ultra long life avionics architecture have not bee addressed by the experiments mentioned above. Examples of these issues are:

1. The wireless interface in this prototype has to handle only a few function blocks, and the arbitration problem is simplified by the host computer. In the next step, a more sophisticated wireless protocol should be developed so that large number of *Generic Function Blocks* should be able to communicate directly with each other simultaneously.
2. A more realistic testbed need to be constructed, in which all sensors have wireless interface and can communicate with any *Generic Function Blocks* directly.
3. A "self-repair" capability should be developed in each Generic Function Block, so that a failed block can be salvaged and reused.
4. The wireless interface among the Generic Function Blocks might interfere with the telecommunication system or other on-board electronics. The effect of the wireless interface need to be investigated and design techniques should be developed to minimize such interference.
5. Traditional verification techniques such as accelerated life test might be too expensive or taking too long to verify the reliability and lifetime of the Ultra Long Life Avionics architecture. New verification techniques have to be developed for systems that are ultra long life.

ACKNOWLEDGEMENT

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

[1] S. Chau and J. Blosiu, "Ultra Long Life System Concept, Rev 1," Internal Document, Jet Propulsion Laboratory, Feb 16, 2001.

[2] A. Sengupta and A. Sen, "On the diagnosability problem for a general model of diagnosable systems", *Information Science*, vol. 42, pp. 83-94, 1987.

[3] A. Sengupta and C. Rhee, "On a generalization of self-implicating structures in diagnosable systems", *IEEE Trans. Circuits and Systems*, vol. 40, no. 4, pp. 239-245, April, 1993.

[4] Y. C. Shin and A. Sengupta, "Self-diagnosability of multiprocessor systems with hybrid faults: diagnosis by comparison approach", *IEEE Trans. Circuits and Systems*, vol. 40, pp. 355-358, May, 1993.

[5] IEEE std 802.11b-1999, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band